Targeted Association Querying for Dynamic and Distributed databases

T. Hamsapriya[1], Dr.S.Sumathi[2]

ABSTRACT

Advances in computing and communication over wired and wireless networks have resulted in many pervasive distributed computing environments. Many of these environments deal with different distributed sources of voluminous data, multiple compute nodes, and distributed user community. Implementation of data mining ideas in high-performance parallel and distributed computing environments is thus becoming crucial for ensuring system scalability and interactivity as data continues to grow inexorably in size and complexity. Recent business trends favor targeted association querying to constrain the search to specific items. This paper discusses a unified approach for distributed datamining and targeted querying that reduces the communication overhead by constructing a distributed itemset tree. The tree also provides the ability for incremental mining and transaction tracing. Construction of the itemset tree has O(N) space and time requirements. Experiments were conducted to observe the behavior of the itemset tree in distributed and incremental mining.

Keywords: Distributed mining, Incremental mining, dynamic datasets, targeted querying, market baskets.

1. INTRODUCTION

An important problem in datamining is discovering association rules from databases of transactions [1] [2] [3] [4] [5], where each transaction contains a set of items. Association mining technique searches for a group of frequently co-occurring items in a market basket type of data, and turns these groups into business-oriented rules. Previous research has focused predominantly on how to obtain exhaustive lists of such associations. However, in most of the applications, users wish to examine a transaction database by giving specialized queries [6]. These queries are called Targeted queries. For instance, one user may want to learn about the buying habits of customers, who frequently purchase milk and fruits. The user may not be interested in a complete list of items, but prefers to restrict the search to specific items. A query of this kind may seek all frequent itemsets containing milk and fruits. Another user may seek all the rules of the form [bread, milk] => s (where s is an itemset), with a confidence of at least 20 percent. In both the cases, repeated search for all itemsets would waste precious computational resources.

Most of the existing mining methods concentrate on determining the frequent sets [2] [3] [4] [5]. So the task of answering targeted association querying requires the generation of exhaustive list of all frequent itemsets. The task of generating the frequent itemsets is quite challenging. The search space is exponential in the number of items occurring in the database. The support threshold limits the output to a hopefully reasonable subspace. In order to reduce the response times, a compact data structure called an itemset tree can be constructed to answer targeted queries, determine frequent itemset and other queries.

[1]Assistant Professor, Department of Computer Science and Engineering

[2]Assistant Professor, Department of Electrical and Electronics Engineering
PSG College of Technology, Coimbatore – 641 004, India

E-mail: prish_67@yahoo.co.in

Most of the databases could be massive, containing millions of transactions, making support counting a tough problem. Also, a user may be interested in generating a global model of the database, thus the sites must exchange some information about their local models. In a distributed scenario the user may be interested in not only knowing the global model of the database, but also the differences between the local models [9]. To expedite the processing of such queries the proposed work converts the market baskets into itemset trees on each site. The advantage of this approach is that it requires only one database scan to construct the tree.

In this article, an efficient parallel and distributed incremental approach is proposed for answering targeted queries on dynamic and distributed datasets. This work here is an extension of that presented in [6]. The main contributions of this paper are:

1. A distributed mining algorithm that minimizes the communication costs for mining over a wide area network, which is used to update a global model.
2. Experimentation and validation on synthetic databases.

The rest of the paper is organized as follows. Section 2 gives a brief review of the related work. Itemset tree construction is discussed in Section 3.

Section 4 discusses the proposed model. Section 5 presents the analysis of the results and finally the paper is concluded in Section 6.

2. Related Work

2.1 Distributed Mining

In recent years, progress in computer technology and data acquisition has led to large distributed databases in fields ranging from banking through biology and astronomy. Often, the size of a dataset or the rate at which data is inserted or removed is so large that existing sequential algorithms are ineffective. In these cases, parallel or distributed algorithms are necessary. In [10], Park and Kargupta give an overview of a wide variety of distributed data mining algorithms for association rule mining, classifier learning, and collective data mining, and clustering, among others. In particular, there has been much research into parallel and distributed algorithms for mining association rules [11] [12][13]. A common approach for mining distributed databases is the *centralized* one, in which all data is moved to a single central location and then mined. Another common approach is the *local* one, where models are built locally in each site, and then moved to a common location where they are combined. The latter approach is the quickest but often the least accurate, while the former approach is more accurate but generally quite expensive in terms of time required. In the search for accurate and efficient solutions, some intermediate approaches have been proposed [13] [14] [15].

These techniques are devised to scale up a given algorithm (e.g., APRIORI, ECLAT, etc.).Data is distributed (or in some cases, replicated) among different sites and a data-mining algorithm is executed in parallel on each site. These approaches do not take into account the possible distributed nature of the data. Some assume a high-speed network environment and perform excessive communication operations. These approaches are not efficient when the databases are distributed over a geographically wide area. The FDM (Fast Distributed Mining) algorithm presented in [14] attempts to cut down on communication between sites by first having each site mine its local frequent itemsets and then exchanging this information to find the global frequent itemsets. In [15], Schuster and Wolff proposed the DISTRIBUTED

DECISION MINER algorithm and several variations, which do not assume that each local frequent itemset is potentially a global frequent itemset. In [16], three strategies of distributed data mining are examined, based on what information is exchanged between sites. The three strategies are those that move results (MR), move models (MM), and move data (MD). The MD strategy is generally avoided, since it can be costly in terms of communication. The authors propose the Papyrus system [16], which makes use of all three strategies to different degrees, based on the values of a cost function and an error function that take into account the cost of transmitting data between nodes and the distribution of data over the nodes.

2.2 Mining Dynamic Databases

Some recent effort has been devoted to the problem of performing incremental data mining in dynamic databases. Parthasarathy and Ramakrishnan propose a parallel incremental method for performing two-dimensional discretization on a dynamic dataset in [17]. A method for incremental sequence mining named ISM is presented in [18]. It is based on the SPADE algorithm [19] for discovering frequent sequences. The incremental algorithm works by keeping track of the maximally frequent and minimally infrequent sequences in the original database. It combines this information with the incremental data to minimize the amount of the original database that need to be re-scanned. Incremental versions of the GSP [21] and MFS [22] frequent sequence-mining algorithms, respectively named GSP+ and MFS+ were presented in [23]. Unlike the ISM algorithm, GSP+ and MFS+ are able to handle both insertions and deletions, and are not limited to a vertical database layout.

Some of these algorithms cope with the problem of determining when to update the current model of frequent itemsets, while others update the model after an arbitrary number of updates. To decide when to update, Lee and Cheung [23] propose the DELI algorithm, which uses statistical sampling methods to determine when the current model is outdated.

3. Itemset Tree

In this section we describe the basic algorithm using an itemset tree for frequent itemset mining and targeted querying. This algorithm is extended to the parallel and distributed mining to generate an accurate global model. Itemset tree is a compact data structure that stores the transactions. It can answer specialized queries and determine the frequent itemsets.

3.1 Construction

Let L denote a set of itemsets and let N be the number of distinct items encountered in L. Each item is identified with an integer from [1, N], so that the items in an itemset, $p = [a_1, a_2, \ldots a_p]$, can be ordered: $a_i < a_j$ for $i < j$, where a_i and a_j are integers identifying i^{th} and j^{th} items, respectively. If the symbols p, r, and s denote itemsets, then

- p is an ancestor of r and can be written as $p \subseteq r$, iff $p = [a_1, a_2, \ldots a_m]$, $r = [a_1, a_2, \ldots a_n]$, and m d" n.
- s is the largest common ancestor of p and r, and write $s \subseteq p \cap r$, iff $s \subseteq p$, $s \subseteq r$, and there is no s ' such that s ' p, $s' \subseteq r$, and $s \subseteq s'$, s "" s'.
- r is a child of p iff $p \subseteq r$ (p is ancestor of r) and there is no s, different from p and r, such that $p \subseteq s \subseteq r$.

Let p and r be itemsets. Then, one and only one out of the following conditions hold:

1. $p \cap r = \emptyset$
2. $p = r$
3. $(p$ "" $r)$ and $(p \subseteq r)$
4. $(p$ "" $r)$ and $(r \subseteq p)$
5. $(p$ "" $r)$ and $(p \cap r = s)$ *such that* s "" p, s "" r.

Proof. Condition 1 represents the case when two itemsets have only the empty set as the common ancestor. If they

have some other common ancestor, then they are either equal to each other (condition 2) or different from each other. If they are different from each other, then one can be an ancestor of the other (conditions 3 and 4) or they have a common ancestor different from either of them (condition 5).

An itemset tree, T, consists of a root and a (possibly empty) set, $\{T_1, T_2, T_k\}$, each element of which is an itemset tree. The root is a pair [s, f(s)], where s is an itemset and f(s) is a frequency. If s_i denotes the itemset associated with the root of the i^{th} subtree, then $s \subseteq s_i$ s "' s_i must be satisfied for all i.

Any itemset tree is a partially ordered set of the pairs [itemset, frequency]. This means that the concrete layout of a root's children is irrelevant as long as they all occupy the same level in the tree. A tree with an empty set of subtrees is called a leaf node. All nodes that are not leaf nodes are internal nodes. The itemset associated with the root may be empty, while the itemsets associated with all other internal and leaf nodes are nonempty.

3.2 Item Set Tree Generation Algorithm

Itemset tree is built incrementally through a series of market basket insertions as shown in Table 1.

Table 1 Algorithm 1 for Incorporating a Market Basket, s, in an Itemset Tree, T

```
Construct {s,T}

Let R=[s_R,f(s_R)] denote the root of T
and let [c_i,f(c_i)] be R's children

If s=s_R then set f(s_R):=f(s_R)+1
else if s is an ancestor of s_R then

a. Create a new root,[l,f(l):=f(s_R)+1];
b. Append to l,two children:[s,f(s):=1] and [s_R,f(s_R)].

else
 a. set f(s_R):=f(s_R)+1
 b. select an action from the following list:

    1) if s_R is the only common ancestor of s with
       any child of R,or if the set of R's children is empty,
       then create a new leaf,[s,f(s):=1], and make it a child of R.

    2) if c_i=s, then f(c_i)=f(c_i) +1.

    3) if s is an ancestor of c_i,then
       insert [s,f(s):=f(c_i)+1] between R and c_i

    4) if c_i is an ancestor of s,then
       let T(c_i) be the sub-tree rooted at [c_i,f(c_i)] ;
       call construct {s,T(c_i)}

    5) if l is the largest common ancestor of c_i and s
       { such that l≠ s,l≠ c_i } and the itemset associated
       with R is an ancestor of l,then:

    i. Disconnect [c_i,f(c_i)] from R;
    ii. Create a new node , [l,f(l):=f(c_i)+1] as a child of R;
    iii Append to l two children: [s,f(s):=1 and [c_i,f(c_i)]
```

That is, the itemset tree is constructed during a single pass through the given set of market baskets. The principle of the algorithm can be summarized by the following procedure:

Let $\mathbf{R} = [s_R, f(s_R)]$ denote the root, let s denote the next market basket to be inserted, and let c_i denote the children of $\mathbf{R}$. Without loss of generality, assume that s_R is an ancestor of s. If s "' s_R, then the following possibilities are considered. In the simplest case, s_R is the only common ancestor of s with any c_i or the set of R's children is empty (case 1).In this event, a new child, $[s, f(s)] = 1$, is appended to R. If s is identical to some c_i (case 2), then the frequency of c_i is incremented by 1. A more instructive situation occurs when s does have a common ancestor with some c_i. Then, one and only one of the conditions 3-5 of Lemma 3.1 are satisfied. If s is an ancestor of some c_i such that s "' c_i (case 3), then s is inserted between the root and c_i, its frequency being set to $f(s) = f(s) + f(c_i)$. If some c_i, such that s "' c_i, is an ancestor of s (case 4), then the algorithm is recursively called for the subtree rooted at c_i. Finally (case 5), if l is the largest common ancestor of s and c_i such that l "' s and l "' c_i, then a node $[l, f(l)=f(c_i)+1]$ is inserted as a new child of R, and s and c_i become the children of l. Whatever be the action, the root's frequency $f(s_R)$, is always incremented by 1.

The algorithm inserting a market basket in an itemset tree is finite. The only time when the recursion is called is when the conditions of case 4 are satisfied. The last time this can happen is when c_i(in the command Construct [s, T(c_i)] is in a leaf node. Since a leaf node does not have any children, the only choice Algorithm 1 has is case 1. This prevents the recursion from going any deeper.

Illustration-I.

Fig.1 shows the sequence of Algorithm 1 processing the market baskets from database, D = {[1, 4], [2, 5], [1, 2, 3, 4, 5], [1, 2, 4], [2, 5], [2, 4]}

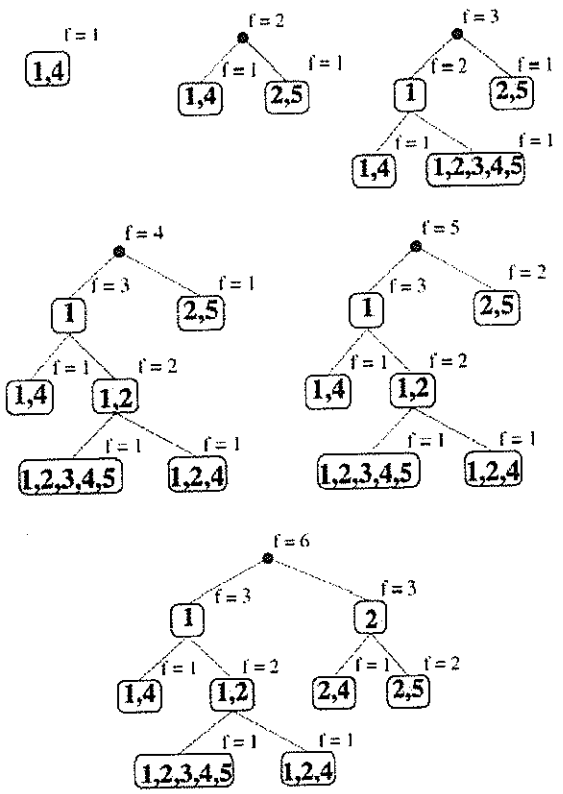


Fig 1 Itemset-tree construction for the database D

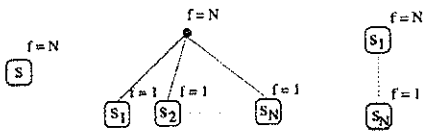
The "size" of an itemset tree is given by the number of nodes (including the root) and by the number of layers (root being the first layer). Simple analysis of Algorithm 1 reveals that no new node is created in case 2; in the event of case 1 or case 3, one new node is created; case 5 creates two new nodes. In case 4, the decision is postponed, but still processing a single basket will never give rise to more than two new nodes and, thus, cannot increase the depth of the tree by more than one layer. This proves the following lemma.

Lemma 1(itemset-tree size). Let T denote the itemset tree generated from a database of N distinct market baskets using Algorithm 1.

1. The number of nodes in T is upper bounded by 2N+1.

2. The number of layers in T is upper bounded by N+1.

This concludes that the memory requirements of the resulting itemset tree are comparable with the size of the original database and, thus, do not represent a major obstacle to employing the technique in real-world domains. Depending on the contents of the database D, three extreme cases, depicted in Fig. 2, are possible. The first case happens when D contains N identical market baskets. In this event, the generated itemset tree consists of a single node, $[s, f(s) = N]$. The second extreme case happens when no two baskets in D have a common ancestor. Then, the tree consists of the root and a single layer with N children. Finally, if the baskets can be ordered in a sequence $s_1 \ldots s_n$, such that s_i is an ancestor of s_{i+1} for any $i \in [1, n-1]$, then each layer will contain a single node.



(a) (b) (c)

Fig 2 Itemset trees: (a) single-child tree (b) single-layer tree (c) linear tree

Lemma 2 (frequency). The frequency of a node in an itemset tree equals the number of market baskets that have passed through this node during the construction of the itemset tree. The frequency of a child is smaller than the frequency of the parent.

Proof. The proof follows immediately from Algorithm 1. Each time a market basket is inserted in the itemset tree, the frequency of each node such that the itemset associated with it is an ancestor of the incoming basket is incremented by 1. The parent's frequency is incremented before a child's frequency is incremented.

3.3 Use of Itemset Trees

Itemset trees can be used to

- Calculate the support of an itemset
- Find all frequent itemsets that contain a user-specified list of items
- Generate all rules that exceed a user-specified minimum confidence and have a user-specified itemset as antecedent.

Table 2 Algorithm 2 that Uses Itemset Tree T to find the Support of Itemsets

Let R denote the root of T and
let $\{c_i, f(c_i)]$ be R's children

Let Ti denote the subtree whose root is $[c_i, f(c_i)]$
To invoke *Count* use f(s):=*Count(s, T, 0)*

Count(s, T, f(s))

For each c_i, such that first_item(c_i) $\sqsubset$ first_item(s):

1. **If** $s \sqsubseteq c_i$ **then return** (f (s)+f(ci))
2. **If** $s \not\sqsubset c_i$ and last_item(c_i) < last_item(s),
   then return (f(s)+ Count(s,T_i,f(s))).

Algorithm 2 sums up the frequencies of all nodes that subsume s (c_i subsumes s, and write $s \subseteq c$, if any item contained in s is also contained in c). At the beginning, the procedure sets $f(s)$, and then examines the itemsets, c_i, associated with the root's children. If the first item of some c_i is greater than the first item in s, then the itemset-tree building algorithm guarantees that none of the nodes of the subtree rooted at c_i subsumes s. This means that the subtree does not affect the support $f(s)$ and can be ignored.

Three different situations can occur in the event of first_item(c_i) d" first_item(s): The first situation is marked by $s \subseteq c_i$, which means that each basket that has passed through this node contained the itemset s. Therefore, it is enough to update the support, $f(s) = f(s) + f(c_i)$, where $f(c_i)$ is the frequency of the child containing c_i.

Second, if $c_i \subseteq s$, the algorithm is applied recursively to the subtree rooted at c_i. Third, if neither condition is satisfied, then none of the nodes in the subtree rooted at c_i can subsume s. In this case, $f(s)$ is not updated. The itemsets are organized in such a way that, to answer an average query, the system will have to visit only a subset of the nodes.

Many nodes will be excluded from further consideration by the conditions first_item(c_i) d" first_item(s) and $c_i \subseteq s$.

Table 3 Algorithm 3 for targeted queries

Let R denote the root of T
and let $[c_i, f(c_i)]$ be R's children

Let T_i denote the subtree whose root is $[c_i, f(c_i)]$

To invoke Selective_mining
use: L:=*Selective_mining*(s,T,{})

Selective_mining(s,T,L)

For each c_i such that first_item(ci) $\leq$ first_item(s):

1. **If** $s \subseteq c_i$ **then**
   i. Let L' be the set of all item sets, l
   such that $s \subseteq l \subseteq c_i$
   ii **return** (L∪L')

2. **If** $s \not\subseteq c_i$ and last_item(c_i)<last_item(s),
   then return (L∪Selective_mining(s,T_i,L))

Algorithm 3 returns the list of all itemsets that subsume the user-specified itemset s and have the support of at least some θ. This algorithm is similar to the one described in Table 2. Both of them constrain their searches by comparisons of the first items in s and c_i. Further on, they both test the subsumption and ancestor relations between

s and c_i. However, there exists some differences. In the event of $s \subseteq c_i$, all itemsets that are subsumed by c_i, and themselves subsume s, have to be appended to the list, **L**, of itemsets subsuming the user-specified items. Moreover, once the search has been completed, two additional steps have to be accomplished. First, some itemsets might appear in **L** more than once and the duplicates have to be merged into a single occurrence (recalculating the support from the tree). Then, itemsets with supports below θ are removed. These steps are listed in the algorithm in Table 4. To answer a query of this kind, the system does not have to search through the entire database. This means that the complexity of this algorithm is upper bounded by **O (N)** where N is the number of market baskets.

Table 4 Algorithm 4 to find the frequent itemset

Let L denote the set of itemsets returned by algorithm 4.
To invoke All_frequent_sets use All_frequent_sets(L, θ)
All_frequent_sets(L, θ)
1. Merge all duplicates found in L(and recalculate the respective supports) 2. Remove from L all itemsets whose supports are less than θ

Itemsets discovered by association mining algorithms are often used to generate rules of the form $s \Rightarrow l$, where s and l are itemsets. An interpretation of a rule like this can be: "60 percent of the customers buying milk and butter will also buy cheese." The percentage (in this case, 60 percent) is sometimes called a confidence in the rule. The technique of itemset trees is being able to support this type of query also.

4. Proposed Work

Let I be a set of distinct attributes called items. Let D be a database of transactions, where each transaction has a unique identifier (tid) and contains a set of items. A set of exactly k items is called a k-itemset. The tidset of an itemset C corresponds to the set of all transaction identifiers (tids) where the itemset C occurs as a subset. The *support count* of C, is the number of transactions of D in which it occurs as a subset. Similarly, the *support* of C, denoted by $\sigma(C)$, is the percentage of transactions of D in which it occurs as a subset. The itemsets that meet a user specified *minimum support* are referred to as *frequent* itemsets. A frequent itemset is *maximal* if it is not subset of any other frequent itemset. The set of all maximal frequent itemsets is denoted as MFI. Targeted querying retrieves the support count of the given itemset S in the transaction set D.

4.1 Itemset mining in distributed datasets

An itemset tree is constructed for the database in each site of the distributed system. The itemset tree is then used to determine the local and the global frequent itemset as well as the targeted queries.

4.1.1Targeted querying in distributed datasets

In a distributed database system, a global frequent itemset must be local frequent in all partitions.

Proof: Let C be an itemset. If $C.sup_i$e" S_D for i= 1,2,...n , then C is a global frequent itemset. Only if it is local frequent in all sites, it can be a global frequent itemset. The local frequent itemsets MFI_is are determined in all sites and are aggregated to form the global frequent itemset.

$$MFI = \bigvee_{i=1}^{n} MFI_i$$

4.1.2 Frequent itemset mining in distributed datasets

The global support count of a targeted itemset C can be obtained by collecting the local support counts and aggregating them. The global support count $C.sup$ of an itemset C is the sum of their local support counts $C.sup_i$.

4.2 Itemset Mining in Dynamic Datasets

In the existing database D, a set of new transactions d^+ is added and a set of old transactions d^- is removed, forming the dynamic dataset D_d *(i.e.* $D_d = (D \vee d^+) - d^-$*)*.

Let S_D be the support of the targeted itemset when mining D. Let e% be the information kept from the current mining that will be used in the next incremental mining operation. Here it consists of itemset trees with partially ordered set of pairs [itemset, frequency] (i.e., all frequent itemsets, along with their support counts, in D). An itemset C is frequent in D_d if σ(C) e" S_D. Note that an itemset C not frequent in D, may become a frequent itemset in D_d .In this case, C is called an *emerged* itemset. If a frequent itemset in D remains frequent in D_d it is called a *retained* itemset.

In this incremental approach, each node consists of a pair *(frequency, database number)* to identify the frequency and the database number. Suppose a node has a frequency *f1* in the database *DB1* and the new frequency for the increment database *DB2* is *f2*, the frequency of the node for the incremented database is *f1+f2*.

5. Results and Discussion

Experiments were conducted with synthetic dataset of varying complexity from IBM Generator on five Pentium 4 nodes with 512MB RAM. The program was written in C.

The database is distributed to all the nodes. The interprocessor communication is implemented using sockets. The average time taken to construct the itemset tree for number of transactions ranging from 10000 through 30000 in 4 sites is shown in Table 5 and Fig 3.

Table 5 Tree construction time in a site

S.No	No. of transactions	Tree Construction time (s)
1	10000	1.181
2	20000	2.207
3	30000	4.42

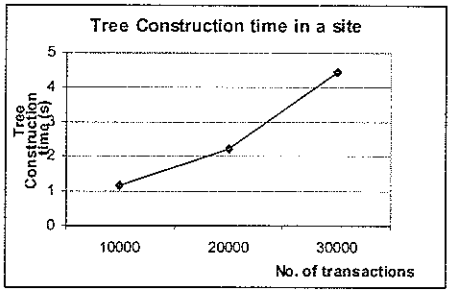


Fig 3 Average Tree Construction time

Time taken for finding the support count of a targeted itemset with 80 items per transaction from an itemset tree constructed with 10000-30000 transactions per node is shown in Table 6 and Fig 4.

Table 6 Response time for targeted querying in distributed datasets

No. of transactions	Response time for targeted querying (s)			
	1	2	3	4
10000	0.563	0.657	0.721	0.812
20000	0.583	0.682	0.743	0.859
30000	0.653	0.702	0.758	0.892

Response time for the Targeted query is the time difference between the time of submission to the time at which the support count is obtained after the construction of the itemset tree.

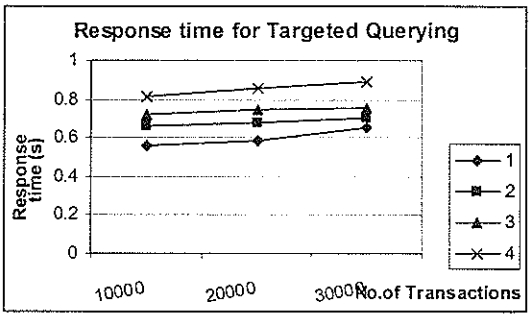


Fig 4 Response time for targeted queries

To study the performance of the itemset tree in dynamic environment, a dataset comprising 10000 transactions was added with the already existing database.

Table 7 Tree construction time for dynamic datasets

Database	No. of transactions	Tree modifiaction time (s)
D1	10000	1..61
D2	20000	2..32
D3	30000	2..85

The new datasets were added to the old database with 30000 transactions. The existing itemset tree was modified to accommodate the new set of transactions. The time taken to modify the existing itemset tree is less than the time taken to construct the entire tree again as shown in Table 7.

Experiments were conducted to determine the response time for targeted querying of an itemset containing 120 unique items in local sites as well as global sites. Table 8 shows the results of targeted querying for an itemset consisting of 120 items in an incremented database.

Table 8 Response time for targeted querying in dynamic datasets

No. of transactions	Response time for targeted querying (s)			
	1	2	3	4
40000	0.923	1.122	1.321	1.426
50000	1.101	1.234	1.421	1.568
60000	1.221	1.342	1.524	1.687

It was interesting to obtain the identity of the targeted query along with the support count. This is due to the fact that each node of the itemset tree stores the frequency and the identity of the dataset to which the itemset belong to. The performance of the itemset tree was evaluated both for addition and deletion of the datasets.

6. Conclusion

In large dynamic and distributed databases, finding the exhaustive list of all the frequent itemsets to answer the user queries is time consuming. This is not suitable for online querying. This project proposes a distributed mining method using itemset tree, which is both space and time efficient compared to existing algorithms. This scheme focuses on mining frequent itemsets and targeted itemsets in distributed dynamic environment. Experimental results show that this scheme also provides the ability for incremental mining and transaction tracing. The itemset tree is order independent and hence it is not dependent on the order in which the market baskets are presented thus making it an ideal candidate for dynamic data mining. This scheme also identifies the database to which the targeted itemset belongs to. Further this approach is applicable to addition as well as deletion of transactions.

7. References

[1] R. Agarwal, C.Agarwal, and V.V.V.Prasad, "Depth-First Generation of Large Itemsets of Association Rules ", IBM Technical Report RC 21538, July 1998.

[2] R. Agarwal, C.Agarwal, and V.V.V.Prasad, "A Tree projection Algorithm for Generation of Frequent Itemsets ", Journal of Parallel and distributed computing,2000.

[3] R. Srikant and R. Agrawal, "Mining Generalized Association Rules," Proc. 21th Int'l Conf. Very Large Databases, pp. 407-419, 1995.

[4] R.J. Bayardo and R. Agrawal, "Mining the Most Interesting Rules," Proc. Fifth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining, pp. 145-154, 1999.

[5] D. Cheung, J. Han, V. Ng, and C. Y. Wong., "Maintenance of discovered association rules in large databases": An incremental updating technique," in *Proc. of the 12th Int'l. Conf. on Data Engineering*, February 1996.

[6] Miroslav Kubat, Alaaeldin Hafez etal, "Itemset trees for Targeted Association Querying", IEEE Transactions on Knowledge and Data Engineering, Vol 15, No.6,November/December 2003,pp 1522-1534.

[7] J. Han and J. Pei, and Y.Yin, "Mining Frequent Patterns without Candidate Generation", *proc* SIGMOD'*00*, 2000.

[8] K. Gouda and M. Zaki, "Efficiently mining maximal frequent itemsets," in *Proc. of the 1*st *IEEE Int'l Conference on Data Mining*, San Jose, USA, November 2001

[9] M. E. Otey, A. Veloso, C. Wang, S. Parthasarathy, and W. Meira Jr., "Mining frequent itemsets in distributed and dynamic databases," in *IEEE International Conference on Data Mining*, 2003.

[10] B.-H. Park and H. Kargupta, "Distributed data mining: Algorithms, systems, and applications," in *Data Mining Handbook*, N. Ye, Ed., 2002.

[11] R. Agrawal and J.C. Shafer, "Parallel Mining of Association Rules," IEEE Trans. Knowledge and Data Eng., vol. 8, no. 6, pp. 962- 969, Dec. 1996.

[12] D.W. Cheung, V.T. Ng, A.W. Fu, and W. Fu, "Efficient Mining of Association Rules in Distributed Databases," IEEE Trans. Knowledge and Data Eng., vol. 8, pp. 911-922, 1996.

[13] A. Veloso, M. E. Otey, S. Parthasarathy, and W. Meira Jr., "Parallel and distributed frequent itemset mining on dynamic datasets," in *International Conference On High Performance Computing*, 2003.

[14] D. Cheung, J. Han, V. Ng, A. Fu, and Y. Fu, "A fast distributed algorithm for mining association rules," in *4th Intl.Conf. Parallel and Distributed Info. Systems*, 1996a.

[15] A. Schuster and R. Wolff, "Communication-efficient distributed mining of association rules" in *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, ACM Press, 2001, pp. 473-484.

[16] R. L. Grossman, S. M. Bailey, H. Sivakumar, and A. L. Turinsky, "Papyrus: A system for data mining over local and wide-area clusters and super-clusters", 1999.

[17] S. Parthasarathy and A. Ramakrishnan, "Parallel incremental 2d discretization," *IEEE International Conference on Parallel and Distributed Processing*, 2002.

[18] S. Parthasarathy, M. J. Zaki, M. Ogihara, and S. Dwarkadas, "Incremental and interactive sequence mining," in *CIKM*, 1999, pp. 251-258.

[19] Zaki, "Efficient enumeration of frequent sequences," in *CIKM: ACM CIKM International Conference on Information and Knowledge Management*. ACM, SIGIR, and SIGMIS, 1998.

[20] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *Proc. 5th Int. Conf. Extending Database Technology, EDBT*, P. M. G. Apers, M. Bouzeghoub, and G. Gardarin, Eds., vol.1057. Springer-Verlag, 25-29 1996, pp. 3-17.

[21] M. Zhang, B. Kao, C. Yip, and D. Cheung, "A GSP-based efficient algorithm for mining frequent sequences," in *Proc.of IC-AI'001, Las Vegas, Nevada, USA*, June 2001.

[22] M. Zhang, B. Kao, D. W.-L. Cheung, and C. L. Yip, "Efficient algorithms for incremental update of frequent sequences," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2002, pp. 186-197.

[23] D. Cheung, S. Lee, and B. Kao, "A general incremental technique for maintaining discovered association rules," in *Proc. of the* 5th *Int'l. Conf. on Database Systems for Advanced Applications*, April 1997, pp. 1-4.

Authors' Biography :

Ms. T. Hamsapriya is an Assistant Professor, Dept. of CSE, PSG College of Technology. She has 10 years of teaching experience and 4 years of industrial experience. She is currently pursuing her research in Parallel and Distributed Computing. She has published many papers in the area of parallel data mining, Evolutionary Computing and Distributed Computing in International and National Journals and Conferences.

Dr. S. Sumathi is an Assistant Professor, Dept. of EEE, PSG College of Technology. She has published many papers in the area of data mining, and Soft Computing in International and National Journals and Conferences. She has authored three books in the area of Data mining and Neural Networks.